David H. Bailey and Robert F. Lucas, Editors

Performance Tuning of Scientific Applications

List of Figures

List of Tables

Contents

Chapter 4

Large-Scale Numerical Simulations on High-End Computational Platforms

Leonid Oliker, Jonathan Carter, Vincent Beckner, John Bell, Harvey Wasseman
CRD/NERSC, Lawrence Berkeley National Laboratory

Mark Adams
APAM Department, Columbia University

Stéphane Ethier
Princeton Plasma Physics Laboratory, Princeton University

Erik Schnetter
CCT, Louisiana State University

4.1 Introduction

After a decade where high-end computing was dominated by the rapid pace of improvements to CPU frequencies, the performance of next-generation

supercomputers is increasingly differentiated by varying interconnect designs and levels of integration. Understanding the tradeoffs of these system designs is a key step towards making effective petascale computing a reality. In this work, we conduct an extensive performance evaluation of five key scientific application areas: micro-turbulence fusion, quantum chromodynamics, micro-finite-element solid mechanics, supernovae and general relativistic astrophysics that use a variety of advanced computation methods, including adaptive mesh refinement, lattice topologies, particle in cell, and unstructured finite elements. Scalability results and analysis are presented on three current high-end HPC systems, the IBM BlueGene/P at Argonne National Laboratory, the Cray XT4 and the Berkeley Laboratory's NERSC Center, and an Intel Xeon cluster at Lawrence Livermore National Laboratory[1]. In this paper, we present each code as a section, where we describe the application, the parallelization strategies, and the primary results on each of the three platforms. Then we follow with a collective analysis of the codes performance and make concluding remarks.

4.2 HPC Platforms and Evaluated Applications

In this section we outline the architecture of the systems used in this study. We have chosen three three architectures which are often deployed at high-performance computing installations, the Cray XT, IBM BG/P, and an Intel/Infiniband cluster. The cpu and node architectures are fully described in Chapter *??*, Section *??* of this book, so we confine ourselves here to a description of the interconnects. In selecting the systems to be benchmarked, we have attempted to cover a wide range of systems having different interconnects. The Cray XT is designed with tightly integrated node and interconnect fabric. Cray have opted to design a custom network ASIC and messaging protocol and couple this with a commodity AMD processor. In contrast, the Intel/IB cluster is assembled from off the shelf high-perfortmance networking components and Intel server processors. The final system, BG/P, is custom designed for processor, node and interconnect with power efficiency as one of the primary goals. Together these represent the most common design tradeoffs in the high performance computing arena. Table 4.1 shows the size and topology of the three system interconnects.

Franklin: Cray XT4: Franklin, a 9660 node Cray XT4 supercomputer, is located at Lawrence Berkeley National Laboratory (LBNL). Each XT4 node contains a quad-core 2.3 GHz AMD Opteron processor, which is tightly integrated to the XT4 interconnect via a Cray SeaStar2+ ASIC through a HyperTransport 2 interface capable of capable of 6.4 GB/s. All the SeaStar routing

[1]The Hyperion cluster used was configured with 45-nanometer Intel Xeon processor 5400 series, which are quad-core "Harpertown nodes."

System Name	Processor Architecture	Interconnect	Total Nodes	MPI BW
Franklin	Opteron	Seastar2+/3D Torus	9660	1.67
Intrepid	BG/P	3D Torus/Fat Tree	40960	1.27
Hyperion	Xeon	Infiniband 4×DDR	576	0.37

TABLE 4.1: Highlights of architectures and file systems for examined platforms. All bandwidths (BW) are in GB/s. "MPI BW" uses unidirection MPI benchmarks to exercise the fabric between nodes with 0.5MB messages.

chips are interconnected in a 3D torus topology with each link is capable of 7.6 GB/s peak bidirectional bandwidth, where each node has a direct link to its six nearest neighbors. Typical MPI latencies will range from 4-8μs, depending on the size of the system and the job placement.

Intrepid: IBM BG/P: Intrepid is a BG/P system located at Argonne National Labs (ANL) with 40 *racks* of 1024 nodes each. Each BG/P node has 4 PowerPC 450 CPUs (0.85 GHz) and 2GB of memory. BG/P implements three high-performance networks: a 3D torus with a peak bandwidth of 0.4 GB/s per link (6 links per node) for point to point messaging; a collectives network for broadcast and reductions with 0.85 GB/s per link (3 links per node); and a network for a low-latency global barrier. Typical MPI latencies will range from 3-10μs, depending on the size of the system and the job placement.

Hyperion: Intel Xeon Cluster: The Hyperion cluster, located at Lawrence Livermore National Laboratory (LLNL), is composed of four scalable units, each consisting of 134 dual-socket nodes utilizing 2.5 GHz quad-core Intel Harpertown processors. The nodes within a scalable unit are fully connected via a 4× IB DDR network with an peak bidirectional bandwidth of 2.0 GB/s. The scalable units are connected together via spine switches providing full bisection bandwidth between scalable units. Typical MPI latencies will range from 2-5μs, depending on the size of the system and the job placement.

The applications chosen as benchmarks come from diverse scientific domains and employ quite different numerical algorithms and are summarized in Table 4.2.

Name	Code Lines	Discipline	Computational Structure
GTC	15,000	Magnetic Fusion	Particle/Grid
OLYMPUS	30,000	Solid Mechanics	Unstructured FE
CARPET	500,000	Relativistic Astrophysics	AMR/Grid
CASTRO	300,000	Compressible Astrophysics	AMR/Grid
MILC	60,000	Quantum Chromodynamics	Lattice

TABLE 4.2: Overview of scientific applications examined in our study

4.3 GTC: Turbulent Transport in Magnetic Fusion

4.3.1 Gyrokinetic Toroidal Code

The Gyrokinetic Toroidal Code (GTC) is a 3D particle-in-cell (PIC) code developed to study the turbulent transport properties of tokamak fusion devices from first principles [41, 42]. The current production version of GTC scales extremely well with the number of particles on the largest systems available. It achieves this by using multiple levels of parallelism: a 1D domain decomposition in the toroidal dimension (long way around the torus geometry), a multi-process particle distribution within each one of these toroidal domains, and a loop-level multitasking implemented with OpenMP directives. The 1D domain decomposition and particle distribution are implemented with MPI using 2 different communicators: a toroidal communicator to move particles from one domain to another, and an intra-domain communicator to gather the contribution of all the particles located in the same domain. Communication involving all the processes is kept to a minimum. In the PIC method, a grid-based field is used to calculate the interaction between the charged particles instead of evaluating the N^2 direct binary Coulomb interactions. This field is evaluated by solving the gyrokinetic Poisson equation [40] using the particles' charge density accumulated on the grid. The basic steps of the PIC method are: *(i)* Accumulate the charge density on the grid from the contribution of each particle to its nearest grid points. *(ii)* Solve the Poisson equation to evaluate the field. *(iii)* Gather the value of the field at the position of the particles. *(iv)* Advance the particles by one time step using the equations of motion ("push" step). The most time-consuming steps are the charge accumulation and particle "push", which account for about 80% to 85% of the time as long as the number of particles per cell per process is about two or greater.

In the original GTC version described above, the local grid within a toroidal domain is replicated on each MPI process within that domain and the particles are randomly distributed to cover that whole domain. The grid work, which comprises of the field solve and field smoothing, is performed redundantly on each of these MPI processes in the domain. Only the particle-related work is fully divided between the processes. This has not been an issue until recently due to the fact that the grid work is small when using a large number of particles per cell. However, when simulating very large fusion devices, such as the international experiment ITER [33], a much larger grid must be used to fully resolve the microturbulence physics, and all the replicated copies of that grid on the processes within a toroidal domain make for a proportionally large memory footprint. With only a small amount of memory left on the system's nodes, only a modest amount of particles per cell per process can fit. This problem is particularly severe on the IBM Blue Gene system where the amount of memory per core is small. Eventually, the grid work starts

dominating the calculation even if a very large number of processor cores is used.

The solution to our non-scalable grid work problem was to add another level of domain decomposition to the existing toroidal decomposition. Although one might think that a fully 3D domain decomposition is the ideal choice, the dynamics of magnetically confined charged particles in tokamaks tells us otherwise. The particle motion is very fast in both the toroidal and poloidal (short way around the torus) directions, but is fairly slow in the radial direction. In the toroidal direction, the domains are large enough that only 10% of the particles on average leave their domain per time step in spite of their high velocities. Poloidal domains would end up being much smaller, leading to a high level of communication due to a larger percentage of particles moving in and out of the domains at each step. Furthermore, the poloidal grid points are not aligned with each other in the radial direction, which makes the delineation of the domains a difficult task. The radial grid, on the other hand, has the advantage of being regularly spaced and easy to split into several domains. The slow average velocity of the particles in that direction insures that only a small percentage of them will move in and out of the domains per time step, which is what we observe.

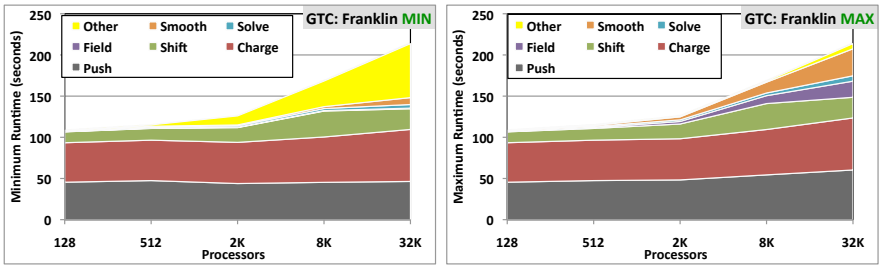
One disadvantage, however, is that the radial width of the domains needs to decrease with the radius in order to keep a uniform number of particles in each domain since the particles are uniformly distributed across the whole volume. This essentially means that each domain will have the same volume but a different number of grid points. For a small grid having a large number of radial domains, it is possible that a domain will fall between two radial grid points. Another disadvantage is that the domains require a fairly large number of ghost cells, from 3 to 8 on each side, depending on the maximum velocity of the particles. This is due to the fact that our particles are not point particles but rather charged "rings", where the radius of the ring corresponds to the Larmor radius of the particle in the magnetic field. We actually follow the guiding center of that ring as it moves about the plasma, and the radius of the ring changes according to the local value of the magnetic field. A particle with a guiding center sitting close to the boundary of its radial domain can have its ring extend several grid points outside of that boundary. We need to take that into account for the charge deposition step since we pick four points on that ring and split the charge between them [40]. As for the field solve for the grid quantities, it is now fully parallel and implemented with the Portable, Extensible Toolkit for Scientific Computation (PETSc) [50].

Overall, the implementation of the radial decomposition in GTC resulted in a dramatic increase in scalability for the grid work and decrease in the memory footprint of each MPI process. We are now capable of carrying out an ITER-size simulation of 130 million grid points and 13 billion particles using 32,768 cores on the BG/P system, with as little as 512 Mbytes per core. This would not have been possible with the old algorithm due to the replication of the local poloidal grid (2 million points).

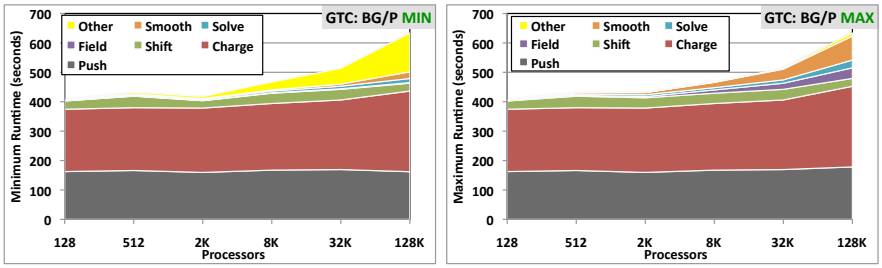
4.4 GTC Performance

Figure 4.1 shows a weak scaling study of the new version of GTC on Franklin, Intrepid, and Hyperion. In contrast with previous scaling studies that were carried out with the production version of GTC and where the computational grid was kept fixed [48, 49], the new radial domain decomposition in GTC allows us to perform a true weak scaling study where both the grid resolution and the particle number are increased proportionally to the number of processor cores. In this study, the 128-core benchmark uses 0.52 million grid points and 52 million particles while the 131,072-core case uses 525 million grid points and 52 billion particles. This spans three orders of magnitude in computational problem size and a range of fusion toroidal devices from a small laboratory experiment of 0.17 m in minor radius to an ITER-size device of 2.7 m, and to even twice that number for the 131,072-core test parameters. A doubling of the minor radius of the torus increases its volume by 8 if the aspect ratio is kept fixed. The Franklin Cray XT4 numbers stop at the ITER-size case on 32,768 cores due to the number of processors available on the system although the same number of cores can easily handle the largest case since the amount of memory per core is much larger than on BG/P. The concurrency on Hyperion stops at 2048, again due to the limited number of cores on this system. It is worth mentioning that we did not use the shared-memory OpenMP parallelism in this study although it is available in GTC.

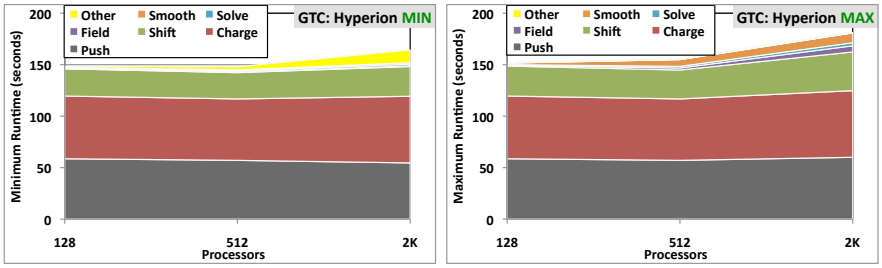
The results of the weak scaling study are presented as area plots of the wall clock times for the main steps in the time-advanced loop as the number of cores increases from 128 to 32,768 in the case of the XT4, from 128 to 131,072 for the BG/P, and from 128 to 2048 for Hyperion. The main steps of the time loop are: accumulating the particles' charge density on the grid ("charge" step, memory scatter), solving the Poisson equation on the grid ("solve" step), smoothing the potential ("smooth" step), evaluating the electric field on the grid ("field" step), advancing the particles by one time step ("push" phase including field gather), and finally, moving the particles between processes ("shift"). The first thing we notice is that the XT4 is faster than the other 2 systems for the same number of cores. It is about 30% faster than Hyperion up to the maximum of 2048 cores available on that system. Compared to BG/P, Franklin is 4 times faster at low core count but that gap decreases to 2.4 times faster at 32,768 cores. This clearly indicates that the new version of GTC scales better on BG/P than Franklin, a conclusion that can be readily inferred visually from the area plots. The scaling on BG/P is impressive and shows the good balance between the processor speed and the network speed. Both the "charge" and "push" steps have excellent scaling on all three systems as can be seen from the nearly constant width of their respective areas on the plots although the "charge" step starts to increase at large processor count. The "shift" step



(a)



(b)



(c)

FIGURE 4.1: GTC weak scaling, showing minimum and maximum times, on the (a) Franklin XT4, (b) Intrepid BG/P, and (c) Hyperion Xeon systems. The total number of particles and grid points are increased proportionally to the number of cores, describing a fusion device the size of ITER at 32,768 cores.

also has very good scaling but the "smooth" and "field" steps account for the

largest degradation in the scaling at high processor counts. They also account for the largest differences between the minimum and maximum times spent by the MPI tasks in the main loop as can be seen by comparing the left (minimum times) and right (maximum times) plots for each system. These two steps hardly show up on the plots for the minimum times while they grow steadily on the plots for the maximum times. They make up for most of the unaccounted time on the minimum time plots, which shows up as "Other." This indicates a growing load imbalance as the number of processor-cores increases. We note that the "push" "charge" and "shift" steps involve almost exclusively particle-related work while "smooth" and "field" involve only grid-related work.

One might conclude that heavy communication is responsible for most of the load imbalance but we think otherwise due to the fact that grid work seems to be the most affected. We believe that the imbalance is due to a large disparity in the number of grid points handled by the different processes at high core count. It is virtually impossible to have the same number of particles and grid points on each core due to the toroidal geometry of the computational volume and the radially decomposed domains. Since we require a uniform density of grid points on the cross-sectional planes, this translates to a constant arc length (and also radial length) separating adjacent grid points, resulting in less points on the radial surfaces near the center of the circular plane compared to the ones near the outer boundary. Furthermore, the four-point average method used for the charge accumulation requires six to eight radial ghost surfaces on each side of the radial zones to accommodate particles with large Larmor radii. For large device sizes, this leads to large differences in the total number of grid points that the processes near the outer boundary have to handle compared to the processes near the center. Since the particle work accounts for 80%–90% of the computational work, as shown by the sum of the "push" "charge" and "shift" steps in the area plots, it is more important to have the same number of particles in each radial domain rather than the same number of grid points. The domain decomposition in its current implementation thus targets a constant average number of particles during the simulation rather than a constant number of grid points since both cannot be achieved simultaneously. It should be said, however, that this decomposition has allowed GTC to simulate dramatically larger fusion devices on BG/P and that the scaling still remains impressive.

The most communication intensive routine in GTC is the "shift" step, which moves the particles between the processes according to their new locations after the time advance step. By looking at the plots of wall clock times for the three systems we immediately see that BG/P has the smallest ratio of time spent in "shift" compared to the total loop time. This translates to the best compute to communication ratio, which is to be expected since BG/P has the slowest processor of the three systems. Hyperion, on the other hand, delivered the highest ratio of time spent in "shift", indicating a network performance not as well balanced to its processor speed than the other two systems.

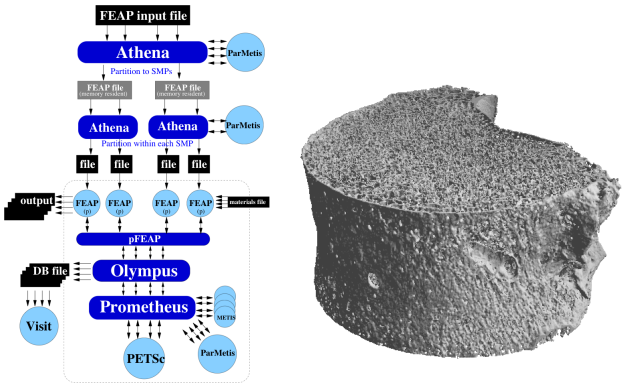
In terms of raw communication performance, the time spent in "shift" on the XT4 is about half of that on the BG/P at low core count. At high processor count, the times are about the same. It is worth noting that on 131,072 cores on BG/P, process placement was used to optimize the communications while this was not yet attempted on Franklin at 32,768 cores.

4.5 OLYMPUS: Unstructured FEM in Solid Mechanics

Olympus is a finite element solid mechanics application that is used for, among other applications, micro-FE bone analysis [10, 18, 43]. Olympus is a general purpose, parallel, unstructured, finite element program and is used to simulate bone mechanics via micro-FE methods. These methods generate finite element meshes from micro-CT data that are composed of voxel elements and accommodate the complex micro architecture of bone in much the same way as pixels are used in a digital image. Olympus is composed of a parallelizing finite element module *Athena*. Athena uses a parallel graph partitioner (ParMetis [34]) to construct a sub-problem on each processor for a serial finite element code *FEAP* [30]. Olympus uses a parallel finite element object *pFEAP*, which is a thin parallelizing layer for FEAP, that primarily maps vector and matrix quantities between the local FEAP grid and the global Olympus grid. Olympus uses the parallel algebraic multigrid linear solver *Prometheus*, which is built on the parallel numerical library *PETSc* [17]. Olympus controls the solution process including an inexact Newton solver and manages the database output (SILO from LLNL) to be read by a visualization application (eg, VISIT from LLNL). Figure 4.2 shows a schematic representation of this system.

The finite element input file is read, in parallel, by Athena which uses ParMetis to partition the finite element graph. Athena generates a complete finite element problem on each processor from this partitioning. The processor sub-problems are designed so that each processor computes all rows of the stiffness matrix and entries of the residual vector associated with vertices that have been partitioned to the processors. This eliminates the need for communication in the finite element operator evaluation at the expense of a small amount of redundant computational work. Given this sub-problem and a small global text file with the material properties, FEAP runs on each processor much as it would in serial mode; in fact FEAP itself has no parallel constructs but only interfaces with Olympus through the pFEAP layer.

Explicit message passing (MPI) is used for performance and portability and all parts of the algorithm have been parallelized for scalability. Hierarchical platforms, such as multi and many core architectures, are the intended target for this project. Faster communication within a node is implicitly exploited by first partitioning the problem onto the nodes, and then recursively calling Athena to construct the subdomain problem for each core. This ap-



(a) (b)

FIGURE 4.2: Olympus code architecture (left), vertebra input CT data (right)

proach implicitly takes advantage of any increase in communication performance within the node, though the numerical kernels (in PETSc) are pure MPI, and provides for good data locality.

4.5.1 Prometheus: parallel algebraic multigrid linear solver

The largest portion of the simulation time for these micro-FE bone problems is spent in the unstructured algebraic multigrid linear solver *Prometheus*. Prometheus is equipped with three multigrid algorithms, has both additive and (true) parallel multiplicative smoother preconditioners (general block and nodal block versions) [9], and several smoothers for the additive preconditioners (Chebyshev polynomials are used in this study [11]).

Prometheus has been optimized for ultra-scalability, including two important features for complex problems using many processors [10]: (1) Prometheus repartitions the coarse grids to maintain load balance and (2) the number of active processors, on the coarsest grids, is reduced to keep a minimum of a few hundred equations per processor. Reducing the number of active processors on coarse grids is all but necessary for complex problems when using tens or hundreds of thousands of cores, and is even useful on a few hundred cores. Repartitioning the coarse grids is important for highly heterogeneous topologies because of severe load imbalances that can result from different coarsening rates in different domains of the problem. Repartitioning is also important on the coarsest grids when the number of processors are reduced

because, in general, the processor domains become fragmented which results in large amounts of communication in the solver.

4.5.2 Olympus Performance

This section investigates the performance of our project with geometrically nonlinear micro-FE bone analyses. The runtime cost of our analyses can be segregated into five primary sections: *(i)* Athena parallel FE mesh partitioner; *(ii)* linear solver *mesh* setup (construction of the coarse grids); *(iii)* FEAP element residual, tangent, and stress calculations; *(iv)* solver *matrix* setup (coarse grid operator construction); and finally *(v)* the solve for the solution.

This study uses a preconditioned conjugate gradient solver—preconditioned with one multigrid V-cycle. The smoothed aggregation multigrid method is used with a second order Chebyshev smoother [11]. Four meshes of this vertebral body are analyzed with discretization scales (h) ranging from 150 microns to 30 microns. Pertinent mesh quantities are shown in Table 4.3. The bone tissue is modeled as a finite deformation (neo-Hookean) isotropic elastic material with a Poisson's ratio of 0.3. All elements are geometrically identical trilinear hexahedra (eight-node bricks). The models are analyzed with displacement control and one solve (i.e., just the first step of the Newton solver). A scaled

h (μm)	150	120	40	30
# degrees of freedom (10^6)	7.4	13.5	237	537
# of elements (10^6)	1.08	2.12	57	135
# of compute cores used	48	184	1696	3840
# of solver iterations (Franklin)	27	38	26	23

TABLE 4.3: Scaled vertebral body mesh quantities. Number of cores used on BG/P is four times greater than shown in this table.

speedup study is conducted on each architecture with the number of cores listed in Table 4.3. Note, the iteration counts vary by as much as 10% in this data due to the non-deterministic nature of the solver, and the fact that the physical domain of each test case is not exactly the same due to noise in the CT data and inexact meshing. In particular, the second test problem (120 μm) always requires a few more iterations to converge.

This weak scaling study is designed to keep approximately the same average number of equations per node/core. Note, a source of growth in the solution time on larger problems is an increase in the number of flops per iteration per fine grid node. This is due to an increase in the average number non-zeros per row which in turn is due to the large ratio of volume to surface area in the vertebral body problems. These vertebral bodies have a large amount of surface area and, thus, the low resolution mesh (150 μm) has a large ratio of surface nodes to interior nodes. As the mesh is refined

the ratio of interior nodes to surface nodes increases, resulting in, on average, more non-zeros per row—from 50 on the smallest version to 68 on the largest. Additionally, the complexity of the construction of the coarse grids in smoothed aggregation has the tendency to increase in fully 3D problems. As this problem is refined, the meshes become more fully 3D in nature—resulting in grids with more non-zeros per row and higher complexities—this would not be noticeable with a fully 3D problem like a cube. Thus, the flop rates are a better guide as to the scalability of the code and scalability of each machine.

Figure 4.3(a) shows the total flop rate of the solve phase and the matrix setup phase (the two parts of the algorithm that are not amortized in a full Newton nonlinear solve) on the Cray XT4 Franklin at NERSC. This shows very good parallel efficiency from 48 to just under 4K cores and much higher flop rates for the matrix setup. The matrix setup uses small dense matrix matrix multiplies (BLAS3) in its kernel which generally run faster than the dense matrix vector multiplies (BLAS2) in the kernel of the solve phase. Figure 4.3(b) shows the times for the major components of a run with one linear solver on the Cray XT4. This data shows, as expected, good scalability (ie, constant times as problem size and processor counts are scaled up).

Figure 4.3(c) shows the flop rates for the solve phase and the matrix setup phase on the Xeon cluster Hyperion at LLNL. Again, we see much higher flop rates for the matrix setup phase and the solve phase is scaling very well up to about 4K processors, but we do observe some degradation the performance of the matrix setup on larger processor counts. This is due to the somewhat complex communication patterns required for the matrix setup as the fine grid matrix (source) and the course grid matrix (product) are partitioned separately and load balancing becomes more challenging because the work per processor is not strictly proportional to the number of vertices on that processor. Figure 4.3(d) shows the times for the major components of one linear solver on Hyperion. This data shows that matrix setup phase is running fast relative to the solve phase (relatively faster than on the Cray) and the AMG setup phase is slowing down on the larger processor counts. Overall the Hyperion data shows that communication fabric is not as good as the Cray's given that tasks with complex communication requirements are not scaling as well on Hyperion as on the XT4.

Figure 4.3(e) shows the flop rates for the solve phase and the matrix setup phase on the IBM BG/P Intrepid at ANL. Figure 4.3(f) shows run times for the solve phase and the matrix setup phase on the BG/P. Note, the scaling study on the BG/P uses four times as many cores as the other two tests, due to lack of memory. Additionally, we were not able to run the largest test case due to lack of memory. These are preliminary results in that we have not addressed some serious performance problems that we are observing on the IBM. First we have observed fairly large load imbalance of about 30%. This imbalance is partially due to the larger amount of parallelism demanded by the BG/P (i.e., we have four times as many MPI processes as on the other systems). We do observe good efficiency in the solve phase but we see significant growth in the

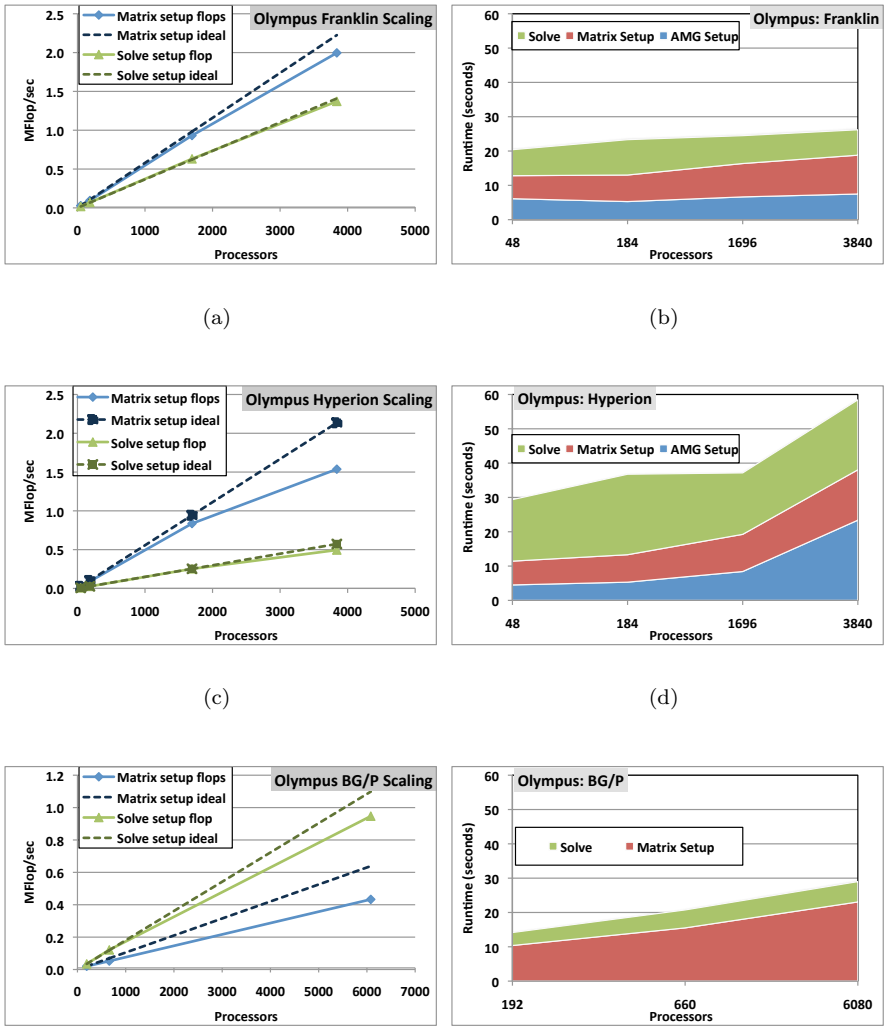


FIGURE 4.3: Olympus weak scaling, flop rates (left), run times (right), for the Cray XT4 (Franklin), Xeon cluster (Hyperion) and IBM BG/P (Intrepid)

matrix setup phase. We speculate that this is due to load imbalance issues. Thus, we are seeing good scalability in the actual solve phase of the solver but we are having problems with load balance and performance on the setup phases of the code. This will be the subject of future investigation.

4.6 Carpet: Higher-Order AMR in Relativistic Astrophysics

4.6.1 The Cactus Software Framework

Cactus [24, 31] is an open-source, modular, and portable programming environment for collaborative HPC computing. It was designed and written specifically to enable scientists and engineers to develop and perform the large-scale simulations needed for modern scientific discovery across a broad range of disciplines. Cactus is used by a wide and growing range of applications, prominently including relativistic astrophysics, but also including quantum gravity, chemical engineering, lattice Boltzmann Methods, econometrics, computational fluid dynamics, and coastal and climate modeling [16, 25, 29, 35, 36, 44, 51, 57]. The influence and success of Cactus in high performance computing was recognized with the IEEE Sidney Fernbach prize, which was awarded to Edward Seidel at Supercomputing 2006.

Among the needs of the Cactus user community have been ease of use, portability, support of large and geographically diverse collaborations, and the ability to handle enormous computing resources, visualization, file I/O, and data management. Cactus must also support the inclusion of legacy code, as well as a range of programming languages. Some of the key strengths of Cactus have been its portability and high performance, which led to it being chosen by Intel to be one of the first scientific applications deployed on the IA64 platform, and Cactus's widespread use for benchmarking computational architectures. For example, a Cactus application was recently benchmarked on the IBM BlueGene/P system at ANL and scaled well up to 131,072 processors.

Cactus is a so-called "tightly coupled" framework; Cactus applications are single executables that are intented to execute within one supercomputing system. Cactus components (called *thorns*) delegate their memory management, parallelism, and I/O to a specialized *driver* component. This architecture enables highly efficient component coupling with virtually no overhead.

The framework itself (called *flesh*) does not implement any significant functionality on its own. It rather offers a set of well-designed APIs which are implemented by other components. The specific set of components which are used to implement these can be decided at run time. For example, these APIs provide coordinate systems, generic interpolation, reduction operations, hyperslabbing, and various I/O methods, and more.

4.6.2 Computational Infrastructure: Mesh Refinement with Carpet

Carpet [6, 55] is an adaptive mesh refinement (AMR) driver for the Cactus framework. Carpet is a driver for Cactus, providing adaptive mesh refinement,

memory management for grid functions, efficient parallelization, and I/O. Carpet provides spatial discretization based on highly efficient block-structured, logically Cartesian grids. It employs a spatial domain decomposition using a hybrid MPI/OpenMP parallelism. Time integration is performed via the recursive Berger–Oliger AMR scheme [21], including subcycling in time. In addition to mesh refinement, Carpet supports multi-patch systems [28, 54, 62] where the domain is covered by multiple, possibly overlapping, distorted, but logically Cartesian grid blocks.

Cactus parallelizes its data structures on distributed memory architectures via spatial domain decomposition, with *ghost zones* added to each MPI process' part of the grid hierarchy. Synchronization is performed automatically, based on declarations for each routine specifying variables it modifies, instead of via explicit calls to communication routines.

Higher order methods require a substantial amount of ghost zones (in our case, three ghost zones for fourth order accurate, upwinding differencing stencils), leading to a significant memory overhead for each MPI process. This can be counteracted by using OpenMP within a multi-core node, which is especially attractive on modern systems with eight or more cores per node; all performance critical parts of Cactus support OpenMP. However, non-uniform memory access (NUMA) systems require care in laying out data structures in memory to achieve good OpenMP performance, and we are therefore using a combination of MPI processes and OpenMP threads on such systems.

We have recently used *Kranc* [3, 32, 39] to generate a new Einstein solver *McLachlan* [23,45]. Kranc is a *Mathematica*-based code generation system that starts from continuum equations in *Mathematica* notation, and automatically generates full Cactus thorns after discretizing the equations. This approach shows a large potential, not only for reducing errors in complex systems of equations, but also for reducing the time to implement new discretization methods such as higher-order finite differencing or curvilinear coordinate systems. It furthermore enables automatic code optimizations at a very high level, using domain-specific knowledge about the system of equations and the discretization method that is not available to the compiler, such as cache or memory hierarchy optimizations or multi-core parallelization. Such optimizations are planned for the future.

Section 4.7 below describes *CASTRO*, an AMR infrastructure using a similar algorithm. We also briefly compare both infrastructures there.

4.6.3 Carpet Benchmark

The Cactus–Carpet benchmark solves the Einstein equations, i.e., the field equations of General Relativity, which describe gravity near compact objects such as neutron stars or black holes. Far away from compact objects, the Einstein equations describe gravitational waves, which are expected to be detected by ground-based and space-based detectors such as LIGO [4], GEO600 [2], and

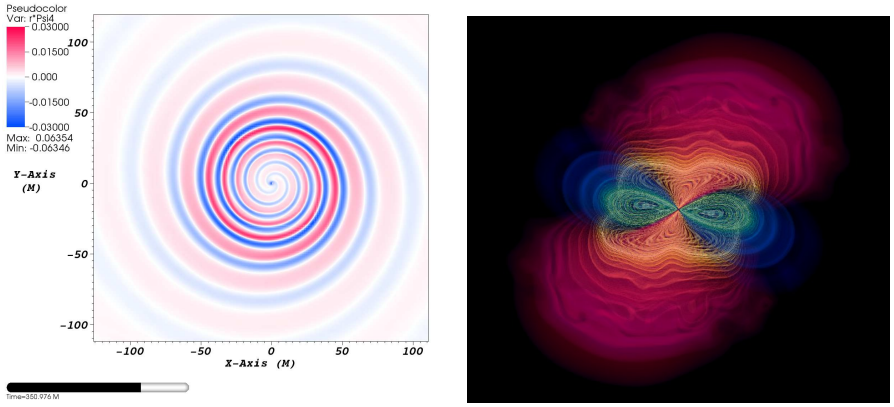
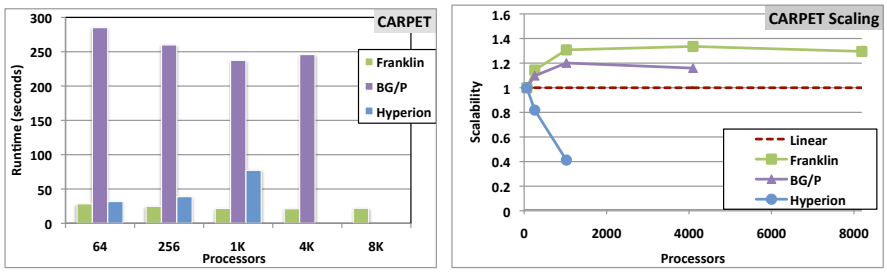


FIGURE 4.4: *Left:* Gravitational radiation emitted during in binary black hole merger, as indicated by the rescaled Weyl scalar $r \cdot \Psi_4$. This simulation was performed with the Cactus-Carpet infrastructure with nine levels of AMR tracking the inspiralling black holes. The black holes are too small to be visible in this figure. [Image credit to Christian Reisswig at the Albert Einstein Institute.] *Right:* Volume rendering of the gravitational radiation during a binary black hole merger, represented by the real part of Weyl scalar $r \cdot \psi_4$. [Image credit to Werner Benger at Louisiana State University.]

LISA [5] in the coming years. This detection will have groundbreaking effects on our understanding, and open a new window onto the universe [59].

We use the BSSN formulation of the Einstein equations as described e.g. in [12,13], which is a set of 25 coupled second-order non-linear partial differential equations. In this benchmark, these equations are discretized using higher order finite differences on a block-structured mesh refinement grid hierarchy. Time integration uses Runge–Kutta type explicit methods. We further assume that there is no matter or electromagnetic radiation present, and use radiative (absorbing) outer boundary conditions. We choose Minkowski (flat spacetime) initial conditions, which has no effect on the run time of the simulations. We use fourth order accurate spatial differencing, Berger–Oliger style mesh refinement [21] with subcycling in time, with higher order order Lagrangian interpolation on the mesh refinement boundaries. We pre-define a fixed mesh refinement hierarchy with nine levels, each containing the same number of grid points. This is a *weak scaling* benchmark where the number of grid points increases with the used number of cores. Figure 4.4 shows gravitational waves from simulated binary black hole systems.

The salient features of this benchmark are thus: Explicit time integration, finite differencing with mesh refinement, many variables (about 1 GByte/core), complex calculations for each grid point (about 5000 flops). This benchmark does *not* use any libraries in its kernel loop such as e.g. BLAS, LA-



(a) (b)

FIGURE 4.5: Carpet benchmark with nine levels of mesh refinement showing(a) weak scaling performance on three examined platforms and (b) scalability relative to concurrency. For ideal scaling the run time should be independent of the number of processors. Carpet shows good scalability except on Hyperion.

PACK, or PETSc, since no efficient high-level libraries exist for stencil-based codes. However, we use automatic code generation [3, 32, 37, 39] which allows some code optimizations and tuning at build time [15].

Our benchmark implementation uses the Cactus software framework [24, 31], the Carpet adaptive mesh refinement infrastructure [6, 54, 55], the Einstein Toolkit [1, 52], and the McLachlan Einstein code [45]. We describe this benchmark in detail in [58].

4.6.4 Carpet Performance

We examined the Carpet weak scaling benchmark described above on several systems with different architectures. Figure 4.5 shows results for the Cray XT4 Franklin, a Blue Gene/P, and Hyperion in terms of runtime and parallel scalability. Overall, Carpet shows excellent weak scalability, achieving superlinear performance on Franklin and Intrepid. On Hyperion scalability breaks down at around 1024 processors, the source of the anomaly is under investigation, as Carpet has demonstrated high scalability up to 12288 processors on other systems with similar architecture, including the Xeon/Infiniband Queen Bee [7] and the AMD/Infiniband Ranger [8] clusters. We were unfortunately not able to obtain interactive access to Hyperion for the Cactus team, and were thus not able to examine this problem in detail — future efforts will focus on identifying the system-specific performance constraints.

We note that we needed to modify the benchmark parameters to run on the BG/P system, since there is not enough memory per core for the standard settings. Instead of assigning AMR components with 25^3 grid points to each

core, we were able to use only 13^3 grid points per component. This reduces the per-core computational load by a factor of about eight, but reduces the memory consumption only by a factor of about four due to the additional inter-processor ghost zones required. Consequently, we expect this smaller version of the benchmark to show less performance and be less scalable.

In some cases (such as Franklin), scalability even improves for very large number of cores. We assume that it is due to the AMR interface conditions in our problem setup. As the number of cores increases, we increase the total problem size to test weak scaling. Increasing the size of the domain by a factor of N increases the number of evolved grid points by N^3, but increases the number of AMR interface points only by a factor of N^2. As the problem size increases, the importance of the AMR interface points (and the associated computation and communication overhead) thus decreases.

4.7 CASTRO: Compressible Astrophysics

CASTRO is a finite volume evolution code for compressible flow in Eulerian coordinates and includes self-gravity and reaction networks. CASTRO incorporates hierarchical block-structured adaptive mesh refinement and supports 3D Cartesian, 2D Cartesian and cylindrical, and 1D Cartesian and spherical coordinates. It is currently used primarily in astrophysical applications, specifically for problems such as the time evolution of Type Ia and core collapse supernovae.

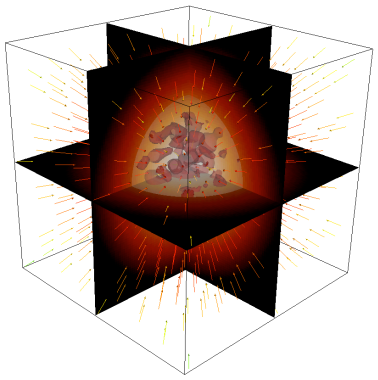
The hydrodynamics in CASTRO is based on the unsplit methodology introduced in [26]. The code has options for the piecewise linear method in [26] and the unsplit piecewise parabolic method (PPM) in [46]. The unsplit PPM has the option to use the less restrictive limiters introduced in [27]. All of the hydrodynamics options are designed to work with a general convex equation of state.

CASTRO supports two different methods for including Newtonian self-gravitational forces. One approach uses a monopole approximation to compute a radial gravity consistent with the mass distribution. The second approach is based on solving the Poisson equation,

$$-\Delta\phi = 4\pi G\rho,$$

for the gravitational field, ϕ. The Poisson equation is discretized using standard finite difference approximations and the resulting linear system is solved using geometric multigrid techniques, specifically V-cycles and red-black Gauss-Seidel relaxation. A third approach in which gravity is externally specified is also available.

Our approach to adaptive refinement in CASTRO uses a nested hierarchy



(a)

Number of Stars	Level1 Grids	Level2 Grids
one	512 (L0) 1728 (L1)	512 (L0) 512 (L1) 1728 (L2)
two	1024 3456	1024 1024 3456
four	2048 6912	2048 2048 6912
eight	4096 13824	4096 4096 13824

(b)

FIGURE 4.6: CASTRO (a) Close-up of a single star in the CASTRO test problem, shown here in red through black on the slice planes are color contours of density. An isosurface of the velocity magnitude is shown in brown, and the vectors represent the radially-inward-pointing self-gravity. (b) Number of grids for each CASTRO test problem.

of logically-rectangular grids with simultaneous refinement of the grids in both space and time. The integration algorithm on the grid hierarchy is a recursive procedure in which coarse grids are advanced in time, fine grids are advanced multiple steps to reach the same time as the coarse grids, and the data at different levels are then synchronized. During the regridding step, increasingly finer grids are recursively embedded in coarse grids until the solution is sufficiently resolved. An error estimation procedure based on user-specified criteria evaluates where additional refinement is needed and grid generation procedures dynamically create or remove rectangular fine grid patches as resolution requirements change.

For pure hydrodynamic problems, synchronization between levels requires only a "reflux" operation in which coarse cells adjacent to the fine grid are modified to reflect the difference between the original coarse-grid flux and the integrated flux from the fine grid. (See [19, 20]). For processes that involve implicit discretizations, the synchronization process is more complex. The basic synchronization paradigm for implicit processes is discussed in [14]. In particular, the synchronization step for the full self-gravity algorithm is similar to the algorithm introduced by [47].

CASTRO uses a general interface to equations of state and thermonuclear

reaction networks, that allows us to easily add more extensive networks to follow detailed nucleosynthesis.

The parallelization strategy for CASTRO is to distribute grids to processors. This provides a natural coarse-grained approach to distributing the computational work. When AMR is used a dynamic load balancing technique is needed to adjust the load. We use both a heuristic knapsack algorithm and a space-filling curve algorithm for load balancing. Criteria based on the ratio of the number of grids at a level to the number of processors dynamically switches between these strategies.

CASTRO is written in C++ and Fortran-90. The time stepping control, memory allocation, file I/O, and dynamic regridding operations occur primarily in C++. Operations on single arrays of data, as well as the entire multigrid solver framework, exist in Fortran-90.

4.7.1 CASTRO and Carpet

The AMR algorithms employed by CASTRO and Carpet (see section 4.6.1 above) share certain features, and we outline the commonalities and differences below.

Both CASTRO and Carpet use nested hierarchies of logically rectangular grids, refining the grids in both space and time. The integration algorithm on the grid hierarchy is a recursive procedure in which coarse grids are first advanced in time, fine grids are then advanced multiple steps to reach the same time as the coarse grids, and the data at different levels are then synchronized. This AMR methodology was introduced by Berger and Oliger (1984) [21] for hyperbolic problems.

Both CASTRO and Carpet support different coordinate systems; CASTRO can be used with Cartesian, cylindrical, or spherical coordinates while Carpet can handle multiple patches with arbitrary coordinate systems [53].

The basic AMR algorithms in both CASTRO and Carpet is independent of the spatial discretization and the time integration methods which are employed. This cleanly separates the formulation of physics from the computational infrastructure which provides the mechanics of gridding and refinement.

The differences between CASTRO and Carpet are mostly historic in origin, coming from the different applications areas that they are or were targeting. CASTRO originates in the hydrodynamics community, whereas Carpet's background is in solving the Einstein equations. This leads to differences in the supported feature set, while the underlying AMR algorithm is very similar. Quite likely both could be extended to support the feature set offered by the respective other infrastructure.

For example, CASTRO offers a generic regridding algorithm based on user-specified criteria (e.g. to track shocks), refluxing to match fluxes on coarse and fine grids, vertex- and cell-centred quantities. These features are well suited to solve flux-conservative hydrodynamics formulations. CASTRO also contains a full Poisson solver for Newtonian gravity.

Carpet, on the other hand, supports features required by the Einstein equations, which have a wave-type nature where conservation is not relevant. Various formulations of the Einstein equations contain second spatial derivatives, requiring a special treatment of refinement boundaries (see [56]). Also, Carpet does not contain a Poisson solver since gravity is already described by the Einstein equations, and no extra elliptic equation needs to be solved.

CASTRO and Carpet are also very similar in terms of their implementation. Both are parallelised using MPI, and support for multi-core architectures via OpenMP has been or is being added.

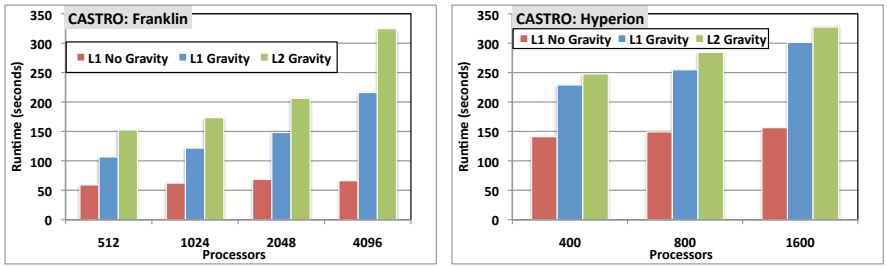
4.7.2 CASTRO Performance

The test problem for the scaling study was that of one or more self-gravitating stars in a 3D domain with outflow boundary conditions on all sides. We used a stellar equation of state as implemented in [60] and initialized the simulation by interpolating data from a 1D model file onto the 3D Cartesian mesh. The model file was generated by a 1D stellar evolution code, Kepler([61]). An initial velocity perturbation was then superimposed onto the otherwise quiescent star — a visualization is shown in Figure 4.6(a).

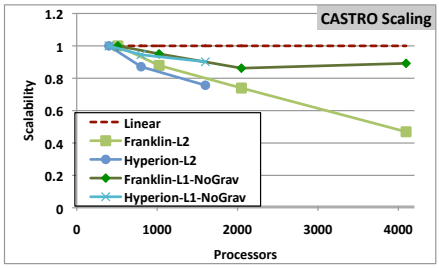
The test problem was constructed to create a weak scaling study, with a single star in the smallest runs, and two, four, and eight stars for the larger runs, with the same grid layout duplicated for each star. Runs on Hyperion were made with one, two, and four stars, using 400, 800, and 1600 processors respectively. Hyperion was in the early testing stage when these runs were made and 1600 processors was the largest available pool at the time, so the smaller runs were set to 400 and 800 processors. Runs on Franklin were made with one, two, four, and eight stars, using 512, 1024, 2048, and 4096 processors respectively. The number of grids for each problem is shown in Table 4.6(b).

Figure 4.7 shows the performance and scaling behavior of CASTRO on Franklin and Hyperion. Results were not collected on Intrepid because the current implementation of CASTRO is optimized for large grids and requires more memory per core than is available on that machine. This will be the subject of future investigation. The runs labeled with "no gravity" do not include a gravity solver, and we note that these scale very well from 512 to 4096 processors. Here *Level1* refers to a simulation with a base level and one level of refinement, and *Level2* refers to a base level and two levels of refinement. Refinement here is by a factor of two between each level. For the *Level2* calculations, the *Level0* base grid size was set to half of the *Level1* calculation's *Level0* base size in each direction to maintian the same effective calculation resolution. Maximum grid size was 64 cells in each direction.

Observe that the runs with "gravity" use the Poisson solver and show less ideal scaling behavior; this is due to the increasing cost of communication in the multigrid solver for solving the Poisson equation. However, the *Level1* and *Level2* calculations show similar scaling, despite that fact that the*Level2* calculation has to do more communication than the *Level1* calculation in order



(a) (b)



(c)

FIGURE 4.7: CASTRO performance behavior with and without the gravity solver using one (L1) and two (L2) levels of adaptivity on (a) Franklin and (b) Hyperion. (c) Shows scalability of the various configurations.

to syncronize the extra level and do more work to keep track of the overhead data required by the extra level. The only exception to this is the highest concurrency Franklin benchmark, where the *Level2* calculation scales noticeably less well than the *Level1*.

Comparing the two architectures, Franklin shows the same or better scaling than Hyperion for the one, two, and four star benchmarks despite using more processors in each case. For the *Level2* calculations with "gravity," even at the lowest concurrency, Franklin is roughly 1.6 times faster than Hyperion, while a performance prediction based on peak flops would be about 1.2.

In a real production calculation, the number and sizes of grids will vary during the run as the underlying data evolves. This changes the calculation's overall memory requirements, communication patterns, and sizes of communicated data, and will therefore effect the overall performance of the entire application Future work will include investigations on optimal gridding, effective utilization of shared memory multicore nodes, communication locality,

reduction of AMR metadata overhead requirements, and the introduction of a hybrid MPI/OpenMP calculation model.

4.8 MILC: Quantum Chromodynamics

The MIMD Lattice Computation (MILC) collaboration has developed a set of codes written in the C language that are used to study quantum chromodynamics (QCD), the theory of the strong interactions of subatomic physics. "Strong interactions" are responsible for binding quarks into protons and neutrons and holding them all together in the atomic nucleus. These codes are designed for use on MIMD (multiple instruction multiple data) parallel platforms, using the MPI library. A particular version of the MILC suite, enabling simulations with conventional dynamical Kogut-Susskind quarks is studied in this paper. The MILC code has been optimized to achieve high efficiency on cache-based superscalar processors. Both ANSI standard C and assembler-based codes for several architectures are provided in the source distribution.[2]

In QCD simulations, space and time are discretized on sites and links of a regular hypercube lattice in four-dimensional space time. Each link between nearest neighbors in this lattice is associated with a 3-dimensional SU(3) complex matrix for a given field [38]. The simulations involve integrating an equation of motion for hundreds or thousands of time steps that requires inverting a large, sparse matrix at each step of the integration. The sparse matrix problem is solved using a conjugate gradient (CG) method, but because the linear system is nearly singular many CG iterations are required for convergence. Within a processor the four-dimensional nature of the problem requires gathers from widely separated locations in memory. The matrix in the linear system being solved contains sets of complex 3-dimensional "link" matrices, one per 4-D lattice link but only links between odd sites and even sites are non-zero. The inversion by CG requires repeated three-dimensional complex matrix-vector multiplications, which reduces to a dot product of three pairs of three-dimensional complex vectors. The code separates the real and imaginary parts, producing six dot product pairs of six-dimensional real vectors. Each such dot product consists of five multiply-add operations and one multiply [22]. The primary parallel programing model for MILC is a 4-D domain decomposition with each MPI process assigned an equal number of sublattices of contiguous sites. In a four-dimensional problem each site has eight nearest neighbors.

Because the MILC benchmarks are intended to illustrate performance achieved during the "steady-state" portion of an extremely long Lattice Gauge

[2] See `http://www.physics.utah.edu/~detar/milc.html` for a further description of MILC.

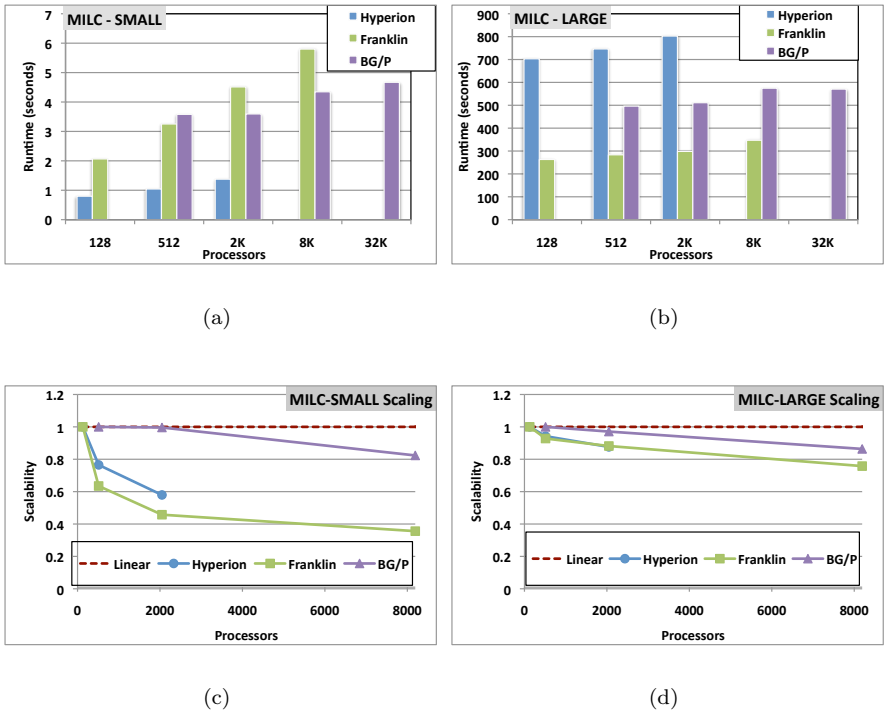


FIGURE 4.8: MILC performance and scalability using (a–b) small and (c–d) large problem configurations.

simulation, each benchmark actually consists of two runs: a short run with a few steps, a large step size, and a loose convergence criterion to let the lattice evolve from a totally ordered state, and a longer run starting with this "primed" lattice, with increased accuracy for the CG solve and a CG iteration count that is a more representative of real runs. Only the time for the latter portion is used.

4.8.1 MILC Performance

We conduct two weak-scaling experiments: a small lattice per core and a larger lattice per core, representing two extremes of how MILC might be used in a production configuration. Benchmark timing results and scalability characteristics are shown in Figures 4.8(a–b) and Figures 4.8(c–d) for small and large problems, respectively. Not surprisingly, scalability is generally better for the larger lattice than for the smaller. This is particularly true for the Franklin XT4 system, for which scalability of the smaller problem is severely

limited. Although the overall computation/communication is very low, one of the main sources of performance degradation is the CG solve. MILC was instrumented to measure the time in five major parts of the code, and on 1024 cores the CG portion is consuming about two-thirds of the runtime. Microkernel benchmarking of the MPI_Allreduce operation on three systems (not shown) shows that the XT4's SeaStar interconnect is considerably slower on this operation at larger core counts. Note, however, that these microkernel data were obtained on a largely dedicated Hyperion system and a dedicated partition of BG/P but in a multi-user environment on the XT4, for which job scheduling is basically random within the interconnect's torus; therefore, the XT4 data likely include the effects of interconnect contention. For the smaller MILC lattice scalability suffers due to the insufficient computation, required to hide this increased Allreduce cost on the XT4.

In contrast, scalability doesn't differ very much between the two lattice sizes on the BG/P system, and indeed, scalability of the BG/P system is generally best of all the systems considered here. Furthermore, it is known that carefully mapping the 4-D MILC decomposition to the BG/P torus network can sometimes improve performance; however, this was not done in these studies and will be the subject of future investigations.

The Hyperion cluster shows similar scaling characteristics to Franklin for the large case, and somewhat better scaling than Franklin for the small.

Turning our attention to absolute performance, for the large case we see that Franklin significantly outperforms the other platforms, with the BG/P system providing the second best performance. For the small case, the order is essentially reversed, with Hyperion having the best performance across the concurrencies studied. MILC is a memory bandwidth-intensive application. Nominally, the component vector and corresponding matrix operations consume 1.45 bytes input/flop and 0.36 bytes output/flop; in practice, we measure for the entire code a computational intensity of about 1.3 and 1.5, respectively, for the two lattice sizes on the Cray XT4. On multicore systems this memory bandwidth dependence leads to significant contention effects within the socket. The large case shows this effect most strongly, with the Intel Harpertown-based Hyperion system lagging behind the other two architectures despite having a higher peak floating-point performance.

4.9 Summary and Conclusions

Computational science is at the dawn of petascale computing capability, with the potential to achieve simulation scale and numerical fidelity at hitherto unattainable levels. However, increasing concerns over power efficiency and the economies of designing and building these large systems are accelerating recent trends towards architectural diversity through new interest in customization

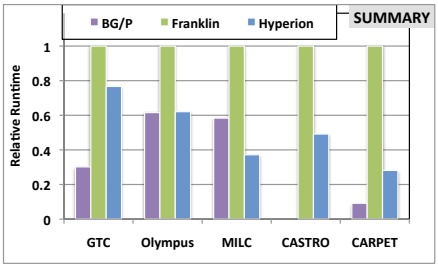


FIGURE 4.9: Results summary for the largest comparable concurrencies of the five evaluated codes on three leading HPC platforms, showing relative runtime performance normalized to fastest system. Note that due to BG/P's restricted memory capacity: Olympus uses 4x the number of BG/P cores, Carpet uses reduced BG/P problem domains, and CASTRO was unable to conduct comparable BG/P experiments.

and tighter system integration on the one hand and incorporating commodity components on the other. Understanding the tradeoffs of these differing design paradigms, in the context of high-end numerical simulations, is a key step towards making effective petascale computing a reality.

In this study, we examined the behavior of a number of key large-scale scientific computations. To maximize the utility for the HPC community, performance was evaluated on the full applications, with real input data and at the scale desired by application scientists in the corresponding domain; these types of investigations require the participation of computational scientists from highly disparate backgrounds. Performance results and analysis were presented on three leading HPC platforms: Franklin XT4, Hyperion Xeon cluster, and Intrepid BG/P, representing some of the most common design trade-offs in the high performance computing arena.

Figure 4.9 presents a summary of the results for largest comparable concurrencies of the five evaluated codes, showing relative runtime performance normalized to fastest system. Observe that the tightly integrated Cray XT system achieves the highest performance, consistently outperforming the Xeon cluster — assembled from commodity components. Comparing to the BG/P is more difficult, as two of the benchmarks use different numbers of processors or different weak scaling parameters. For GTC and MILC, the two directly comparable benchmarks — in one case the Xeon platform outperforms the BG/P platform, but in the other the situation is reversed. For MILC, the low memory bandwidth of the Xeon Clovertown and the relatively poor performance of collective communication of IB as compared to the custom interconnect of BG/P means that BG/P comes out ahead. For the higher computational intensity GTC, the Intel/IB cluster dominates. In the case of Olympus, we can compare across architectures based on the idea that the differences in

number of processors used is representative of how a simulation would be run in practice. Here BG/P and the Xeon cluster have comparable performance, with the XT ahead. For Carpet, the smaller domain sizes used for the BG/P runs makes accurate comparison impossible, but one can certainly say that absolute performance is very poor compared with both the XT and Xeon cluster.

However, as the comparison in Figure 4.9 is at relatively small concurrencies due to the size of the Intel/IB cluster, this is only part of the picture. At higher concurrencies the scalability of BG/P exceeds that of the XT for GTC and MILC, and substantially similar scaling is seen for Olympus. However, for Carpet, XT scalability is better than on BG/P, although it should be noted that both scale superlinearly. While both BG/P and XT have custom interconnects, BG/P isolates portions of the interconnect (called partitions) for particular jobs much more effectively than on the XT, where nodes in a job can be scattered across the torus intermixed with other jobs that are competing for link bandwidth. This is one of the likely reasons for the poorer scalability of some applications on the XT.

Overall, these extensive performance evaluations are an important step toward effectively conducting simulations at the petascale level and beyond, by providing computational scientists and system designers with critical information on how well numerical methods perform across state-of-the-art parallel systems. Future work will explore a wider set of computational methods, with a focus on irregular and unstructured algorithms, while investigating a broader set of HEC platforms, including the latest generation of multi-core technologies.

4.10 Acknowledgements

We thank Bob Walkup and Jun Doi of IBM for the optimized version of MILC for the BG/P system and Steven Gottlieb of Indiana University for many helpful discussions related to MILC benchmarking. We also kindly thank Brent Gorda of LLNL for access to the Hyperion system. This work was supported by the Advanced Scientific Computing Research Office in the DOE Office of Science under contract number DE-AC02-05CH11231. The GTC work was supportred by the DOE Office of Fusions Energy Science under contract number DE-AC02-09CH11466. This work was also supported by the NSF awards 0701566 *XiRel* and 0721915 *Alpaca*, by the LONI *numrel* allocation, and by the NSF TeraGrid allocations TG-MCA02N014 and TG-ASC090007. This research used resources of NERSC at LBNL and ALCF at ANL which are supported by the Office of Science of the DOE under Contract No. DE-AC02-05CH11231 and DE-AC02-06CH11357 respectively.

Bibliography

[1] CactusEinstein toolkit home page. URL `http://www.cactuscode.org/Community/NumericalRelativity/`.

[2] GEO 600. URL `http://www.geo600.uni-hannover.de/`.

[3] Kranc: Automated code generation. URL `http://numrel.aei.mpg.de/Research/Kranc/`.

[4] LIGO: Laser Interferometer Gravitational wave Observatory. URL `http://www.ligo.caltech.edu/`.

[5] LISA: Laser Interferometer Space Antenna. URL `http://lisa.nasa.gov/`.

[6] Mesh refinement with Carpet. URL `http://www.carpetcode.org/`.

[7] Queen Bee, the core supercomputer of LONI. URL `http://www.loni.org/systems/system.php?system=QueenBee`.

[8] Sun Constellation Linux Cluster: Ranger. URL `http://www.tacc.utexas.edu/resources/hpc/`.

[9] M. F. Adams. A distributed memory unstructured Gauss–Seidel algorithm for multigrid smoothers. In *ACM/IEEE Proceedings of SC2001: High Performance Networking and Computing*, Denver, Colorado, November 2001.

[10] M. F. Adams, H.H. Bayraktar, T.M. Keaveny, and P. Papadopoulos. Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In *ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing*, 2004.

[11] M. F. Adams, M. Brezina, J.J Hu, and R. S. Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss–Seidel. *J. Comp. Phys.*, 188(2):593–610, 2003.

[12] Miguel Alcubierre, Bernd Brügmann, Peter Diener, Michael Koppitz, Denis Pollney, Edward Seidel, and Ryoji Takahashi. Gauge conditions for long-term numerical black hole evolutions without excision. *Phys. Rev. D*, 67:084023, 2003.

[13] Miguel Alcubierre, Bernd Brügmann, Thomas Dramlitsch, José A. Font, Philippos Papadopoulos, Edward Seidel, Nikolaos Stergioulas, and Ryoji Takahashi. Towards a stable numerical evolution of strongly gravitating systems in general relativity: The conformal treatments. *Phys. Rev. D*, 62:044034, 2000.

[14] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142:1–46, 1998.

[15] Alpaca: Cactus tools for application-level profiling and correctness analysis. URL `http://www.cct.lsu.edu/~eschnett/Alpaca/`.

[16] Astrophysics Simulation Collaboratory (ASC) home page. URL `http://www.ascportal.org`.

[17] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[18] H. H. Bayraktar, M.F. Adams, P.F. Hoffmann, D.C. Lee, A. Gupta, P. Papadopoulos, and T.M. Keaveny. Micromechanics of the human vertebral body. In *Transactions of the Orthopaedic Research Society*, volume 29, page 1129, San Francisco, 2004.

[19] J. Bell, M. Berger, J. Saltzman, and M. Welcome. A three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Statist. Comput.*, 15(1):127–138, 1994.

[20] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, May 1989.

[21] Marsha J. Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.

[22] C. Bernard, C. DeTar, S. Gottlieb, U.M. Heller, J. Hetrick, N. Ishizuka, L. Kärkkäinen, S.R. Lantz, K. Rummukainen, R. Sugar, D. Toussaint, and M. Wingate. Lattice QCD on the IBM scalable POWERParallel systems SP2. In *ACM/IEEE Proceedings of SC 1995: High Performance Networking and Computing*, San Diego, California, November 1995.

[23] David Brown, Peter Diener, Olivier Sarbach, Erik Schnetter, and Manuel Tiglio. Turduckening black holes: an analytical and computational study. *Phys. Rev. D (submitted)*, 2008. URL `http://arxiv.org/abs/0809.3533`.

[24] Cactus Computational Toolkit home page, URL `http://www.cactuscode.org/`.

[25] Karen Camarda, Yuan He, and Kenneth A. Bishop. A parallel chemical reactor simulation using Cactus. In *Proceedings of Linux Clusters: The HPC Revolution, NCSA*, 2001. URL `http://www.cactuscode.org/Articles/Camarda01.doc`.

[26] P. Colella. Multidimensional Upwind Methods for Hyperbolic Conservation Laws. *Journal of Computational Physics*, 87:171–200, 1990.

[27] P. Colella and M. D. Sekora. A limiter for PPM that preserves accuracy at smooth extrema. *Journal of Computational Physics*, submitted.

[28] Peter Diener, Ernst Nils Dorband, Erik Schnetter, and Manuel Tiglio. Optimized high-order derivative and dissipation operators satisfying summation by parts, and applications in three-dimensional multi-block evolutions. *J. Sci. Comput.*, 32:109–145, 2007.

[29] Fokke Dijkstra and Aad van der Steen. Integration of Two Ocean Models. In *Special Issue of Concurrency and Computation, Practice & Experience*, volume 18, pages 193–202. Wiley, 2005.

[30] FEAP. URL `http://www.ce.berkeley.edu/projects/feap/`.

[31] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[32] Sascha Husa, Ian Hinder, and Christiane Lechner. Kranc: a Mathematica application to generate numerical codes for tensorial evolution equations. *Comput. Phys. Comm.*, 174:983–1004, 2006.

[33] ITER: International thermonuclear experimental reactor. URL `http://www.iter.org/`.

[34] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *ACM/IEEE Proceedings of SC1996: High Performance Networking and Computing*, 1996.

[35] Jong G. Kim and Hyoung W. Park. Advanced simulation technique for modeling multiphase fluid flow in porous media. In *Computational Science and Its Applications - Iccsa 2004, LNCS 2004, by A. Lagana et. al.*, pages 1–9, 2004.

[36] Soon-Heum Ko, Kum Won Cho, Young Duk Song, Young Gyun Kim, Jeong su Na, and Chongam Kim. *Lecture Notes in Computer Science: Advances in Grid Computing - EGC 2005: European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers*, chapter Development of Cactus Driver for CFD Analyses in the Grid Computing Environment, pages 771–777. Springer, 2005.

[37] Kranc: Automated code generation. URL `http://www.cct.lsu.edu/~eschnett/Kranc/`.

[38] A.S. Kronfeld. Quantum chromodynamics with advanced computing. *J. Phys.: Conf. Ser.*, 125:012067, 2008.

[39] Christiane Lechner, Dana Alic, and Sascha Husa. From tensor equations to numerical code — computer algebra tools for numerical relativity. In *SYNASC 2004 — 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania*, 2004. URL `http://arxiv.org/abs/cs.SC/0411063`.

[40] W. W. Lee. Gyrokinetic particle simulation model. *J. Comp. Phys.*, 72:243–269, 1987.

[41] Z. Lin, S. Ethier, T.S. Hahm, and W.M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett.*, 88, 2002.

[42] Z. Lin, T. S. Hahm, W. W. Lee, et al. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science*, 281:1835–1837, 1998.

[43] X.S. Liu, X.H. Zhang, K.K. Sekhon, M.F. Adam, D.J. McMahon, E. Shane, J.P. Bilezikian, and X.E. Guo. High-resolution peripheral quantitative computed tomography can assess microstructural and mechanical properties of human distal tibial bone. *Journal of Bone and Mineral Research*, in press.

[44] Seth Major, David Rideout, and Sumati Surya. Spatial hypersurfaces in causal set cosmology. *Classical Quantum Gravity*, 23:4743–4752, Jun 2006.

[45] McLachlan, a public BSSN code. URL `http://www.cct.lsu.edu/~eschnett/McLachlan/`.

[46] G. H. Miller and P. Colella. A Conservative Three-Dimensional Eulerian Method for Coupled Solid-Fluid Shock Capturing. *Journal of Computational Physics*, 183:26–82, 2002.

[47] F. Miniati and P. Colella. Block structured adaptive mesh and time refinement for hybrid, hyperbolic + n-body systems. *Journal of Computational Physics*, 227:400–430, 2007.

[48] L. Oliker, A. Canning, J. Carter, et al. Scientific Application Performance on Candidate PetaScale Platforms. In *IPDPS:International Conference on Parallel and Distributed Computing Systems*, Long Beach, CA, 2007.

[49] L. Oliker, A. Canning, J. Carter, and J. Shalf. Scientific computations on modern parallel vector systems. In *Proc. SC2004*, Pittsburgh, PA, 2004.

[50] PETSc: Portable, extensible toolkit for scientific computation. URL `http://www.mcs.anl.gov/petsc/`.

[51] D. Rideout and S. Zohren. Evidence for an entropy bound from fundamentally discrete gravity. *Classical Quantum Gravity*, 2006.

[52] Erik Schnetter. Multi-physics coupling of Einstein and hydrodynamics evolution: A case study of the Einstein Toolkit. CBHPC 2008 (Component-Based High Performance Computing) (accepted), 2008.

[53] Erik Schnetter, Peter Diener, Ernst Nils Dorband, and Manuel Tiglio. A multi-block infrastructure for three-dimensional time-dependent numerical relativity. *Classical Quantum Gravity*, 23:S553–S578, 2006. URL `http://arxiv.org/abs/gr-qc/0602104`.

[54] Erik Schnetter, Peter Diener, Nils Dorband, and Manuel Tiglio. A multi-block infrastructure for three-dimensional time-dependent numerical relativity. *Classical Quantum Gravity*, 23:S553–S578, 2006. URL `http://stacks.iop.org/CQG/23/S553`.

[55] Erik Schnetter, Scott H. Hawley, and Ian Hawke. Evolutions in 3D numerical relativity using fixed mesh refinement. *Classical Quantum Gravity*, 21(6):1465–1488, 21 March 2004.

[56] Erik Schnetter, Scott H. Hawley, and Ian Hawke. Evolutions in 3d numerical relativity using fixed mesh refinement. *Classical Quantum Gravity*, 21:1465–1488, 2004. URL `http://arxiv.org/abs/gr-qc/0310042`.

[57] B. Talbot, S. Zhou, and G. Higgins. Review of the Cactus framework: Software engineering support of the third round of scientific grand challenge investigations, task 4 report - earth system modeling framework survey. URL `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20020069014_2002111115.pdf`.

[58] Jian Tao, Gabrielle Allen, Ian Hinder, Erik Schnetter, and Yosef Zlochower. XiRel: Standard benchmarks for numerical relativity codes using Cactus and Carpet. Technical Report CCT-TR-2008-5, Louisiana State University, 2008. URL `http://www.cct.lsu.edu/CCT-TR/CCT-TR-2008-5`.

[59] K. S. Thorne. Gravitational Radiation – a New Window Onto the Universe. (Karl Schwarzschild Lecture 1996). *Reviews of Modern Astronomy*, 10:1–28, 1997.

[60] Frank X. Timmes and F. Douglas Swesty. The accuracy, consistency, and speed of an electron-positron equation of state based on table interpolation of the helmholtz free energy. *"Astrophysical Journal, Supplement"*, 126:501–516, 2000.

[61] T. A. Weaver, G. B. Zimmerman, and S. E. Woosley. Presupernova evolution of massive stars. *Astrophysical Journal*, 225:1021–1029, 1978.

[62] Burkhard Zink, Erik Schnetter, and Manuel Tiglio. Multipatch methods in general relativistic astrophysics – hydrodynamical flows on fixed backgrounds. *Phys. Rev. D*, 77:103015, 2008.